Tema 8. El patrón MVVM

Desarrollo de Interfaces
DAM – IES Doctor Balmis

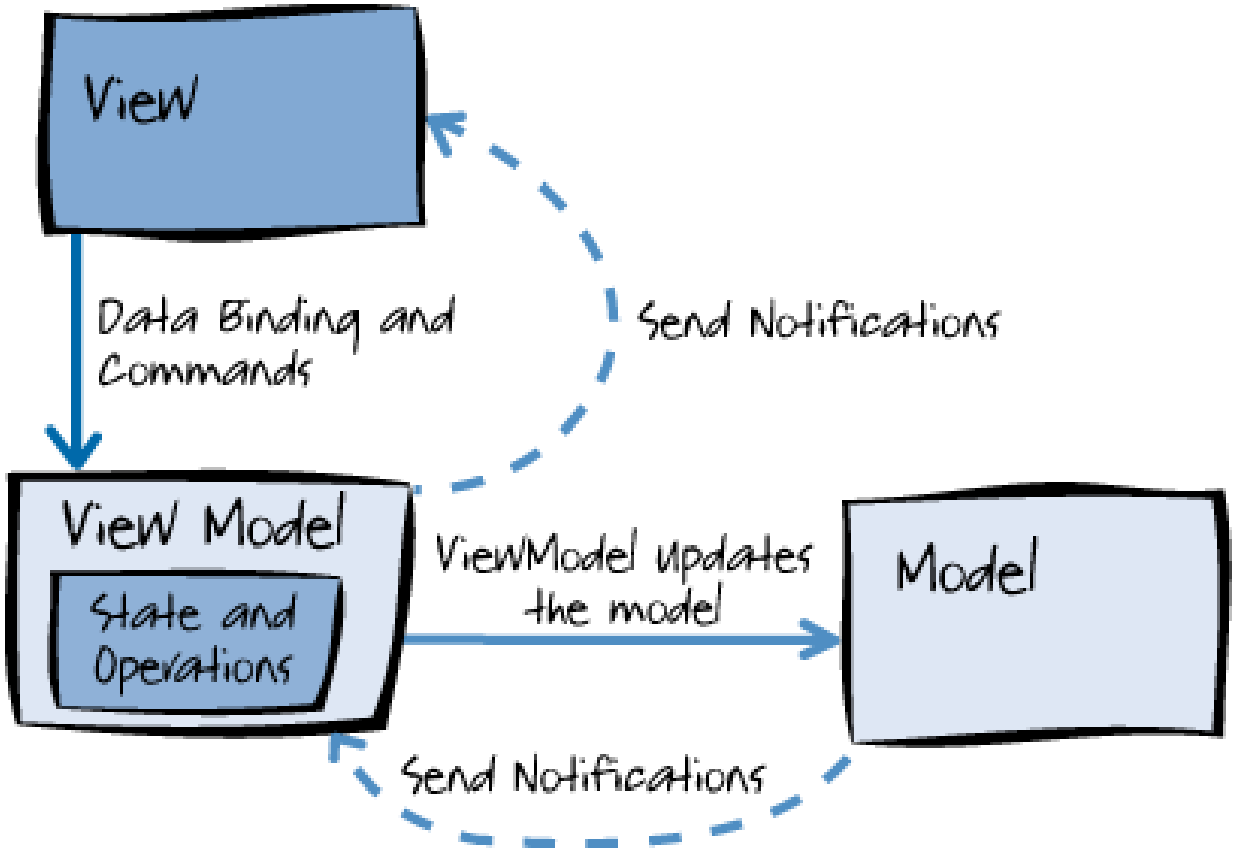
Javier Catalá

INDICE

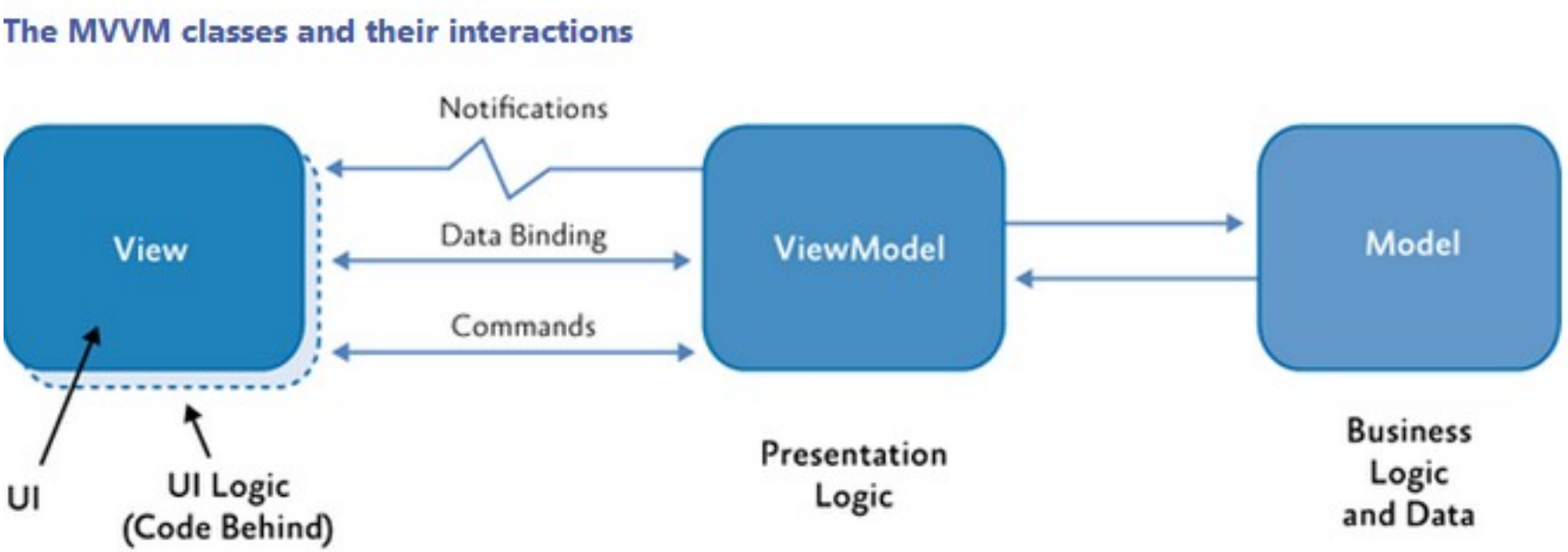
1. El patrón MVVM

- Surge en el año 2005, un año después de WPF
- Propuesto por John Grossman (Microsoft)
- Se presenta como una variación de MVC para WPF
- Similar al patrón *Presentation Model* de Martin Fowler
- En la actualidad se aplica en otras tecnologías (Xamarin, Android, iOS, Javascript...)

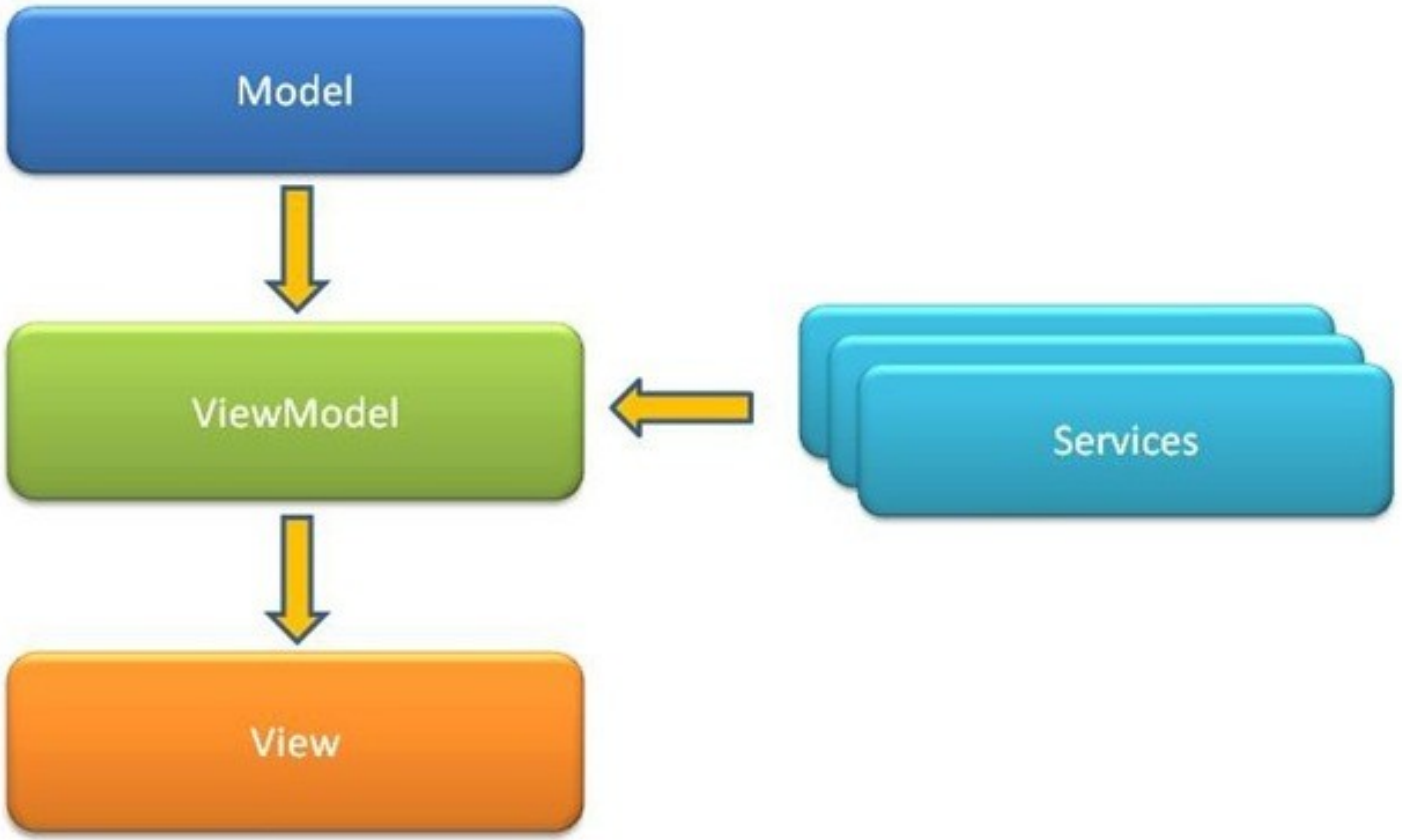
1. El patrón MVVM



1. El patrón MVVM



1. El patrón MVVM



2. Vista (View)

- Define la interfaz de usuario
- Se compone fundamentalmente de XAML
- Puede tener asociado algo de código trasero relacionado directamente con la interfaz
- Suele tener la forma de ventana o de *User Control*
- Se relaciona con la Vista Modelo mediante *bindings* y comandos.

3. Modelo (Model)

- Formado por las clases que representan el dominio de la aplicación
- Puede contener lógica de negocio de la aplicación
- Es actualizado por la Vista Modelo
- Informa a la Vista Modelo de los cambios

4. Vista Modelo (ViewModel)

- Representa una visión abstracta de la vista
- Mantiene el estado de la vista
- Expone propiedades a las que la vista se enlaza
- Contiene las acciones que se invocan desde la vista
- Responsable de actualizar el modelo
- Normalmente existe una por cada vista
- Puede hacer uso de servicios (acceso a BBDD, API Rest,...)

5. Ventajas del patrón MVVM

- Separación de responsabilidades
- Facilita la pruebas unitarias
- Mejora el mantenimiento del código
- Reparto de trabajo más sencillo entre diseñadores y desarrolladores
- Mejora la reutilización del código
- Las ventajas son más visibles en proyectos complejos

6. Ejemplo de MVVM - Modelo

```
class Persona : INotifyPropertyChanged
{
    public double Altura { get; set; }

    public double Peso { get; set; }

    public double Imc
    {
        get { return (Altura == 0) ? 0 : Peso / Altura; }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

6. Ejemplo de MVVM - Vista

```
<Window x:Class="SimpleMVVM.MainWindow"
...
    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.New"
                        Executed="LimpiarCommand_Executed">
        </CommandBinding>
    </Window.CommandBindings>
    <StackPanel DataContext="{Binding Usuario}">
        <TextBlock>Altura (metros)</TextBlock>
        <TextBox Text="{Binding Altura}"></TextBox>
        <TextBlock>Peso (kilos)</TextBlock>
        <TextBox Text="{Binding Peso}"></TextBox>
        <TextBlock>Imc</TextBlock>
        <TextBox Text="{Binding Imc, Mode=OneWay}"></TextBox>
        <Button Command="ApplicationCommands.New">Limpiar</Button>
    </StackPanel>
</Window>
```

6. Ejemplo de MVVM - Vista

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = new MainWindowVM();
    }

    private void LimpiarCommand_Executed(object sender, ExecutedRoutedEventArgs e)
    {
        (this.DataContext as MainWindowVM).Limpiar();
    }
}
```

6. Ejemplo de MVVM – Vista Modelo

```
class MainWindowVM : INotifyPropertyChanged
{
    public Persona Usuario { get; set; }

    public MainWindowVM()
    {
        Usuario = new Persona();
    }

    public void Limpiar()
    {
        Usuario = new Persona();
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```

7. Frameworks MVVM para WPF

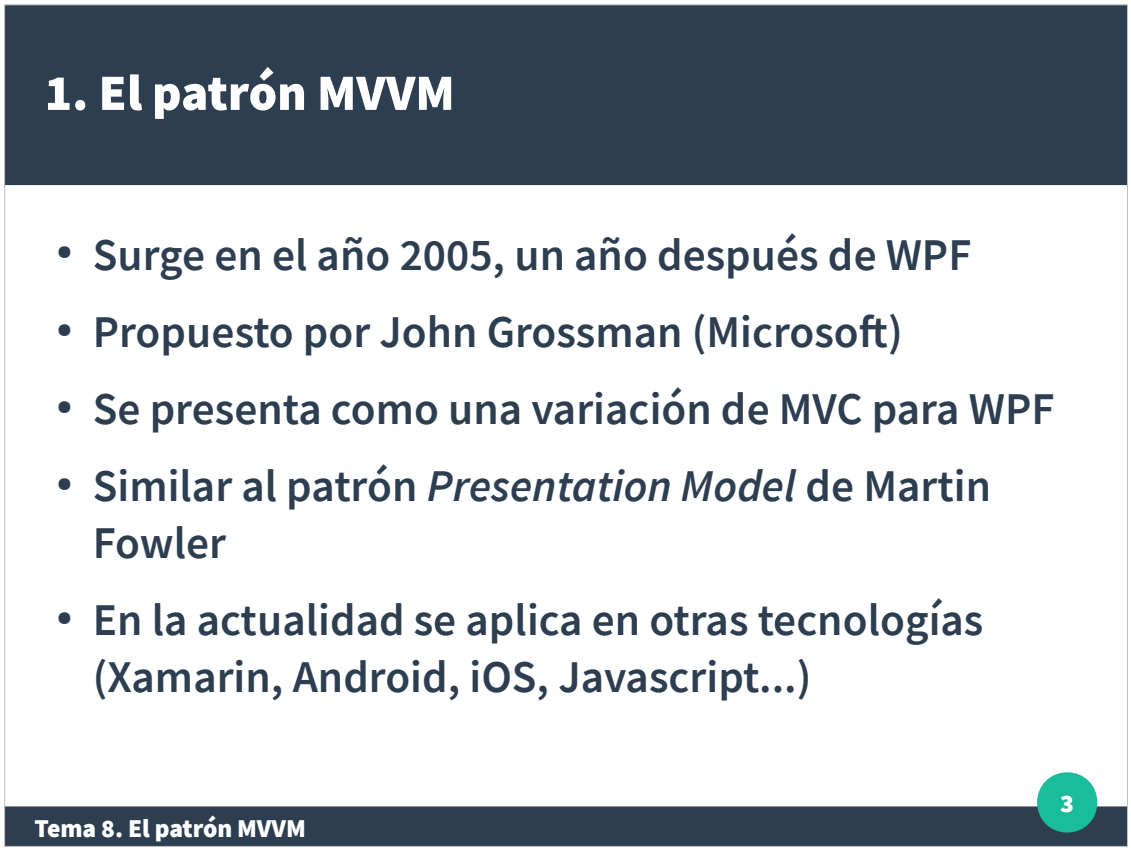
Caliburn.Micro Xaml made easy

Tema 8. El patrón MVVM

Desarrollo de Interfaces
DAM – IES Doctor Balmis

Javier Catalá

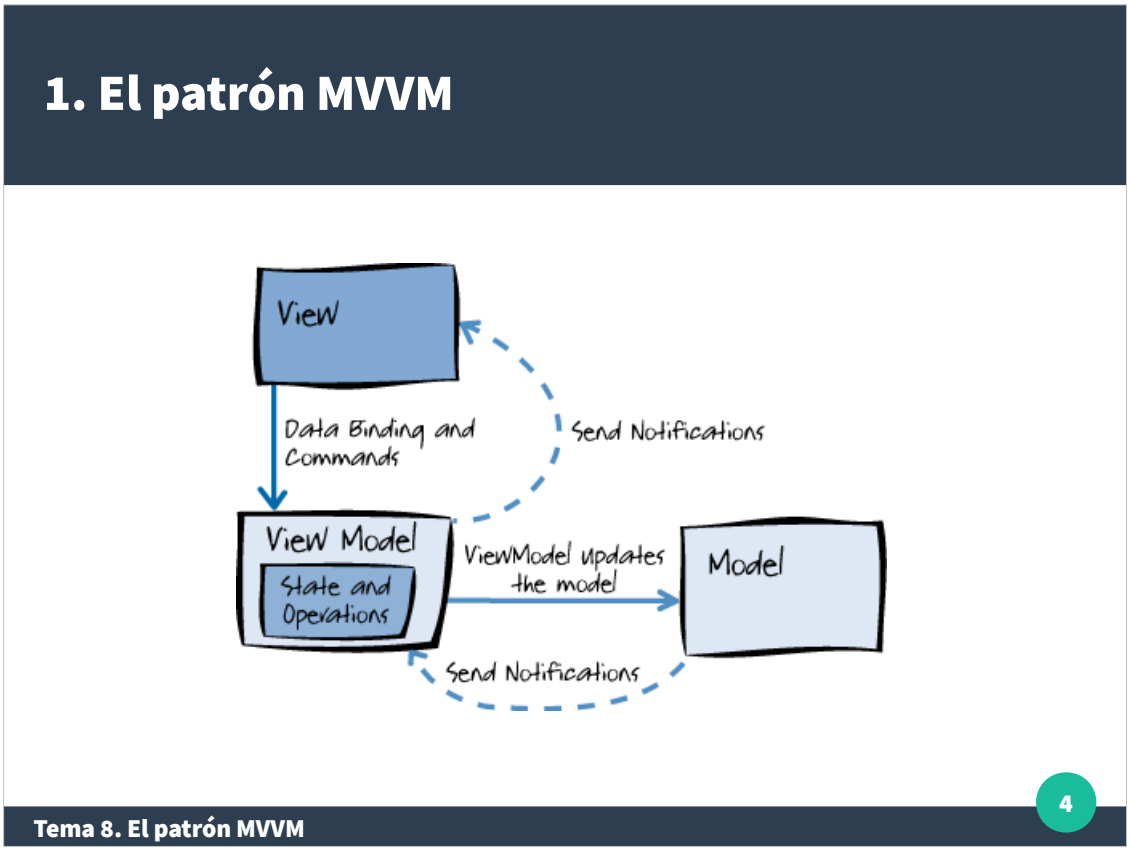
INDICE



El patrón MVVM (*Model-View-ViewModel*) surgión en el año 2005, y fue propuesto por John Grossman, ingeniero de Microsoft.

El patrón surge como una adaptación del patrón MVC (*Model-View-Controller*) a la tecnología WPF, aunque en realidad es más cercano al patrón *Presentation Model* propuesto por el conocido autor Martin Fowler. En este patrón, Fowler proponía separar en una clase el comportamiento y el estado de la presentación, independientemente de los controles usados en la interfaz.

Actualmente, MVVM no se utiliza solo en WPF, si no que se ha extendido a otras tecnologías de presentación como Xamarin, Android, iOS, Javascript,...



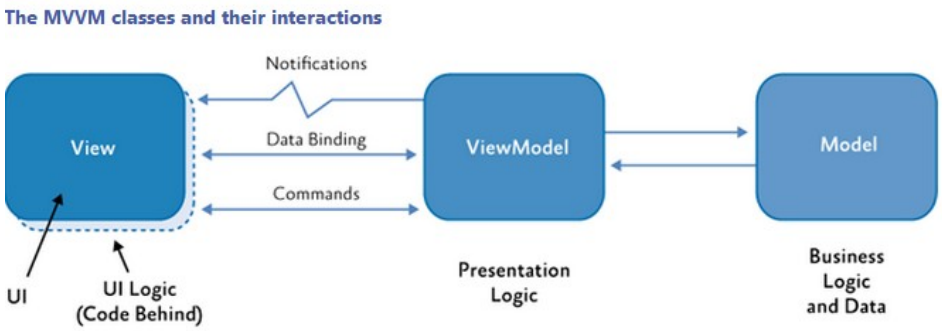
Esta imagen representa los tres componentes del patrón (View-Vista, View Model-Vista Modelo y Model-Modelo) y la relación entre ellos.

La Vista representa lo que se muestra al usuario, y se enlazará utilizando bindings a propiedades de la Vista Modelo. Además, se utilizarán comandos para invocar acciones implementadas en la Vista Modelo.

La Vista Modelo, contiene el estado de la vista y la implementación de las acciones que se llevarán a cabo. Utilizando las interfaces vistas en el curso, notificará a la vista de los cambios en el estado.

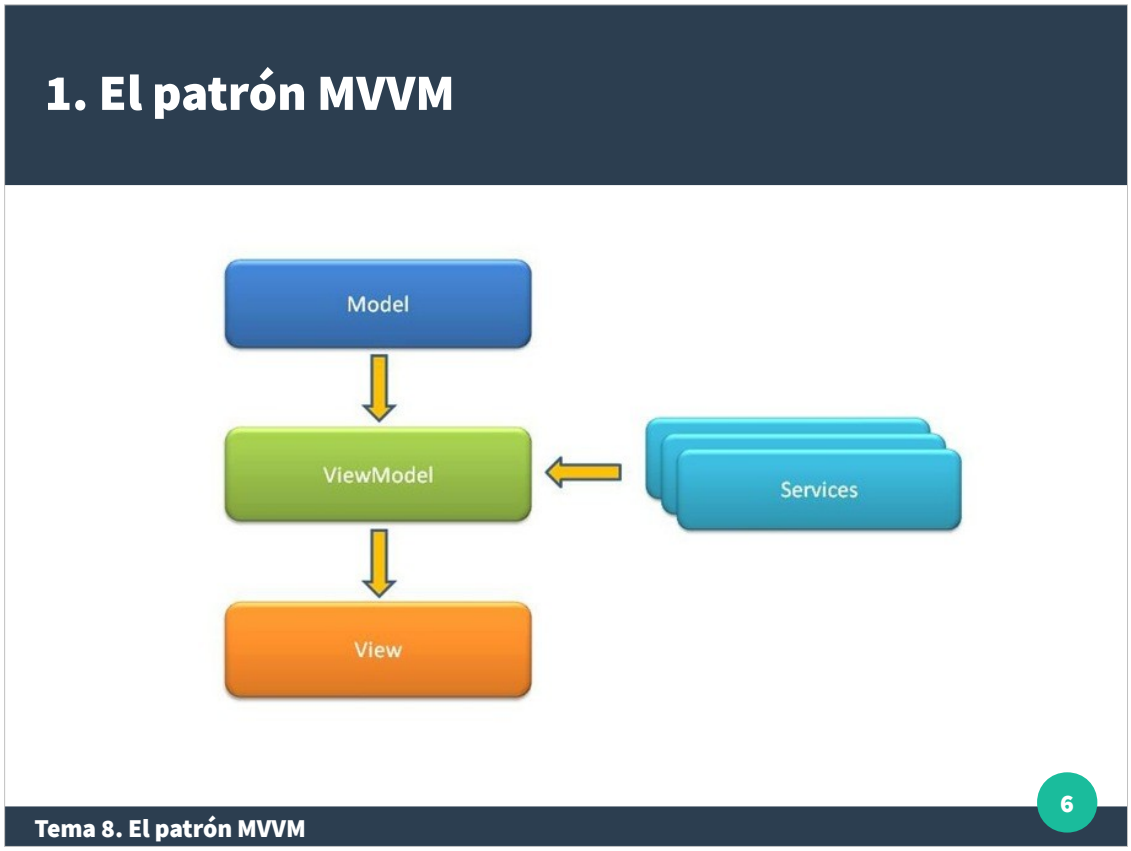
Por último, el Modelo representa el dominio de la aplicación. Es actualizado por la Vista Modelo, y también deberá notificar a la Vista Modelo de sus cambios.

1. El patrón MVVM



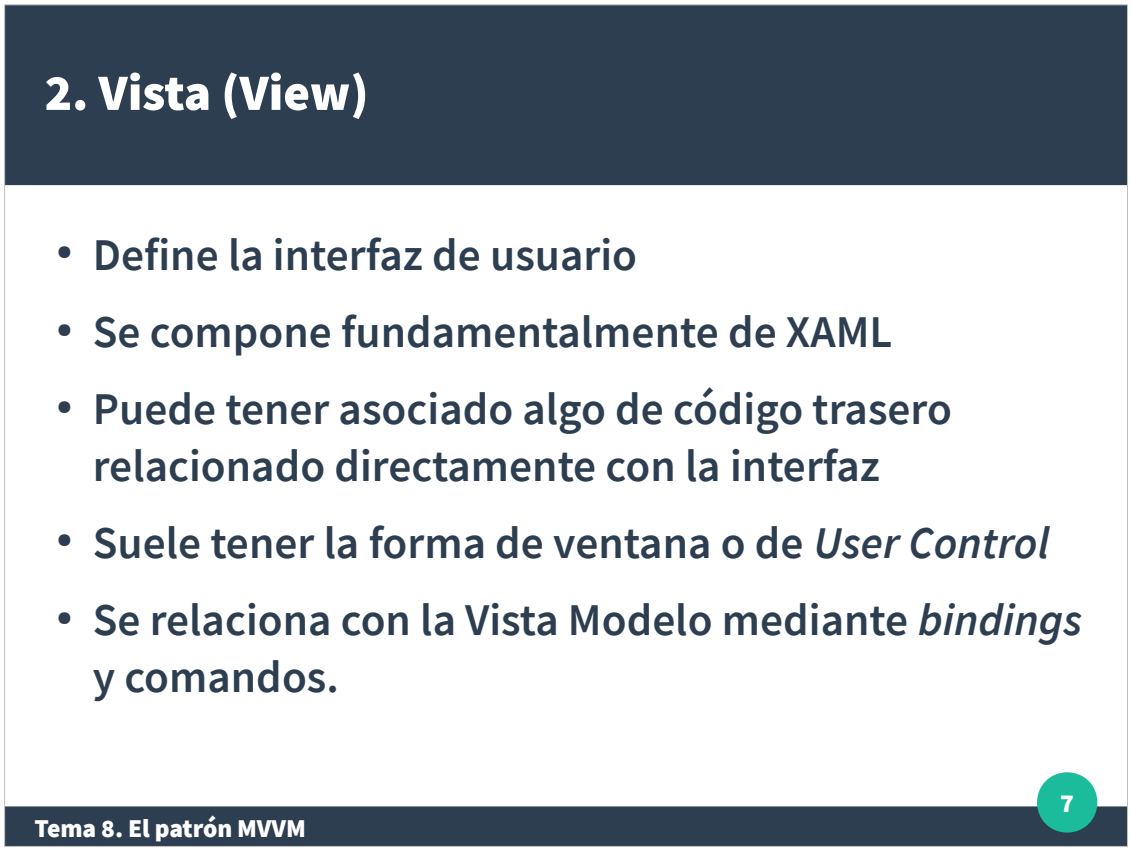
En esta diapositiva se muestra otro esquema del patrón MVVM. Aquí podemos ver como la Vista se corresponde con la interfaz de usuario, la Vista Modelo con la lógica de presentación y el Modelo con la lógica de negocio y los datos.

Centrándonos en la vista, es importante destacar que ésta incluirá fundamentalmente la definición de la interfaz de usuario (XAML), pero que también puede incluir código relacionado directamente con la lógica de la interfaz (code behind de la vista). Pero es importante incidir en que dicho código no debería modificar el estado de la vista ni realizar ninguna acción sobre los datos. En muchas ocasiones, el code behind de la Vista será muy reducido.



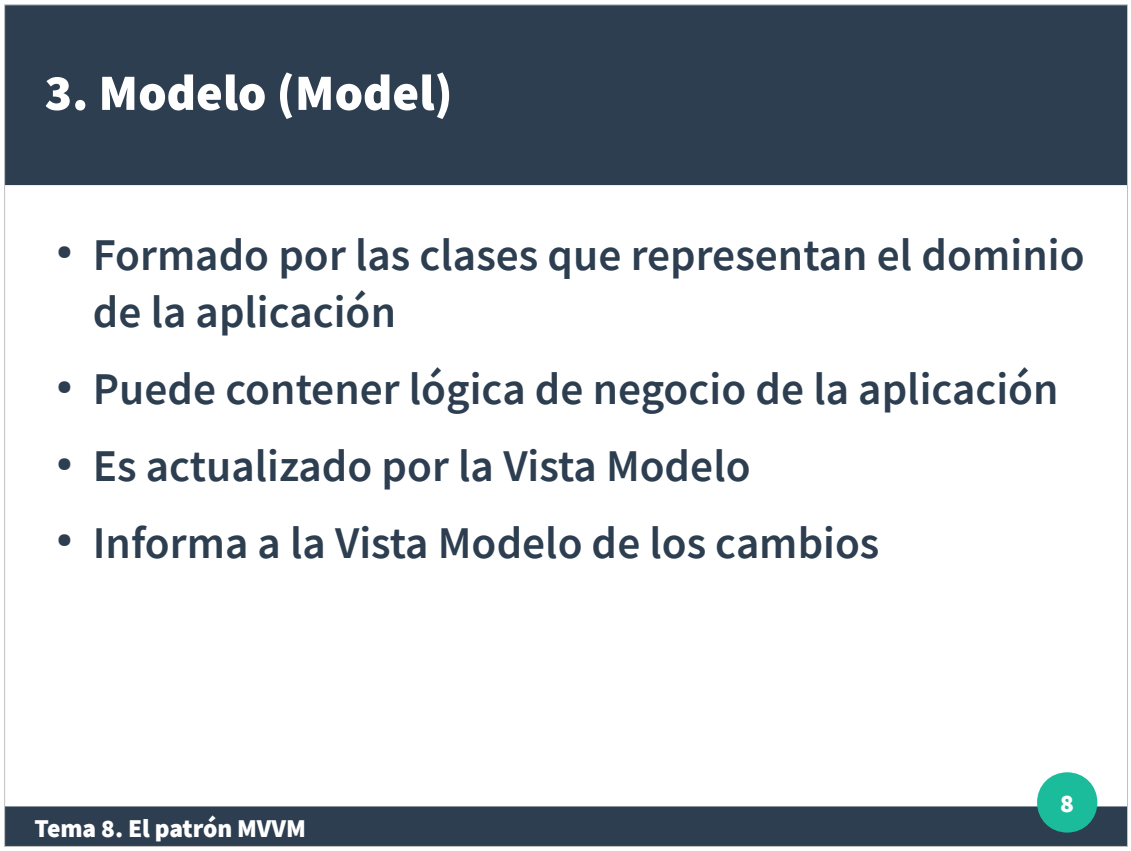
En este esquema del patrón se añade un nuevo elemento, que aunque no forma parte del patrón en sí es muy común encontrarlo en las aplicaciones MVVM. Se trata de los servicios.

Los servicios serán clases utilizadas por la Vista Modelo para realizar acciones que no estén relacionadas con la lógica de presentación. Por ejemplo, será habitual tener un servicio para comunicarse con una base de datos o para hacer uso de una API Rest.



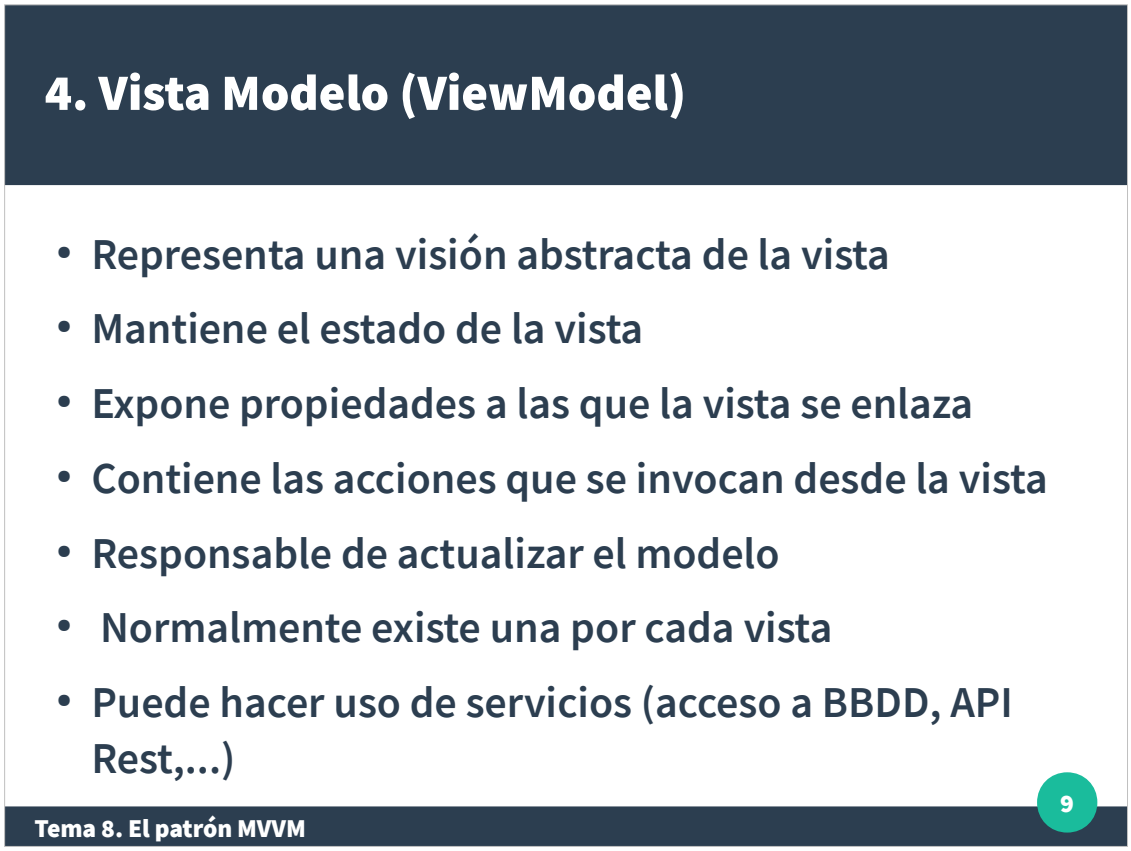
En las siguientes diapositivas se resumen las características principales de cada componente del patrón. Para la Vista serían las siguientes:

- La responsabilidad de la Vista es definir la interfaz de usuario
- En nuestras aplicaciones, estarán compuestas fundamentalmente de XAML, aunque podría incluir algo de código trasero siempre que esté relacionado con la lógica de la interfaz.
- Las Vistas pueden ser ventanas o controles de usuario, de forma que una ventana podría estar compuesta de varias vistas.
- En las Vistas utilizaremos bindings a propiedades de la Vista Modelo, e invocaremos sus comandos.



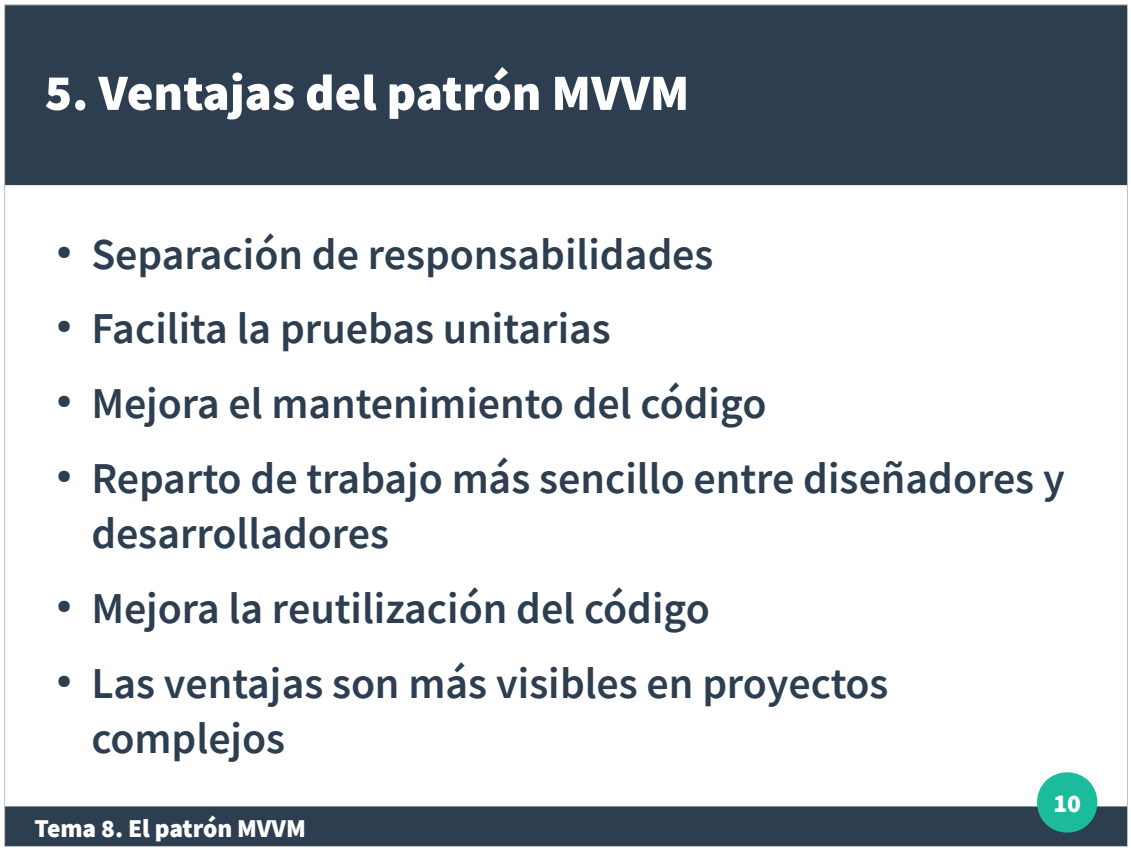
En cuanto al Modelo, podemos destacar las siguientes características:

- Lo componen las clases que forman parte del dominio de la aplicación.
- El Modelo contiene fundamentalmente los datos sobre los que se basa la aplicación, aunque también puede incluir lógica de negocio.
- La Vista Modelo es responsable de actualizar el Modelo.
- El Modelo debe implementar las interfaces necesarias para informar a la Vista Modelo de los cambios.



La Vista Modelo es el elemento central del patrón, y sus principales características son:

- Es una representación abstracta de la Vista.
- Es responsable de mantener el estado de la Vista, y de implementar las acciones que se invocarán desde ella.
- Debe exponer propiedades públicas a las que la Vista se enlazará, y notificará de los cambios en dichas propiedades.
- Suele existir una Vista Modelo por cada Vista.
- Será la responsable de actualizar el Modelo.
- Como ya hemos comentado, puede hacer uso de uno o más servicios (como acceso a BBDD o API Rest) para llevar a cabo su función.



En esta diapositiva se muestran las diferentes ventajas que aporta el uso del patrón MVVM:

- Separación de las responsabilidades de la aplicación entre los diferentes componentes del patrón
- Facilita la realización de pruebas unitarias, que se centrarán en la Vista Modelo
- Se mejora el mantenimiento del código, tanto correctivo como evolutivo.
- Se simplifica el reparto de trabajo entre diseñadores (centrados en la Vista) y programadores (responsables del resto de componentes).
- Facilita la reutilización de código.

Todas estas ventajas se hacen más patentes en proyectos grandes.

6. Ejemplo de MVVM - Modelo

```
class Persona : INotifyPropertyChanged
{
    public double Altura { get; set; }

    public double Peso { get; set; }

    public double Imc
    {
        get { return (Altura == 0) ? 0 : Peso / Altura; }
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```


Vamos a ver un sencillo ejemplo de aplicación MVVM. Se trata de una aplicación para calcular el índice de masa corporal (IMC) de una persona.

En primer lugar se muestra el Modelo. En este caso es una clase que representa a una persona, con propiedades para su altura y peso, y una propiedad de solo lectura para obtener su IMC. La fórmula del IMC es muy sencilla, basta dividir el peso por la altura.

Como se puede observar, el Modelo hace referencia al dominio de la aplicación. Y, además de los datos referentes a dicho dominio (altura y peso) puede contener lógica de negocio (el cálculo del IMC).

6. Ejemplo de MVVM - Vista

```
<Window x:Class="SimpleMVVM.MainWindow"
...
    <Window.CommandBindings>
        <CommandBinding Command="ApplicationCommands.New"
                        Executed="LimpiarCommand_Executed">
        </CommandBinding>
    </Window.CommandBindings>
    <StackPanel DataContext="{Binding Usuario}">
        <TextBlock>Altura (metros)</TextBlock>
        <TextBox Text="{Binding Altura}"></TextBox>
        <TextBlock>Peso (kilos)</TextBlock>
        <TextBox Text="{Binding Peso}"></TextBox>
        <TextBlock>Imc</TextBlock>
        <TextBox Text="{Binding Imc, Mode=OneWay}"></TextBox>
        <Button Command="ApplicationCommands.New">Limpiar</Button>
    </StackPanel>
</Window>
```


La Vista representa lo que verá el usuario. Estará compuesta fundamentalmente de XAML (incluido en esta diapositiva) y algo de code behind (incluido en la diapositiva siguiente).

Vemos que la Vista contiene un *StackPanel* con los campos correspondientes a la altura, el peso y el IMC. El *DataContext* del panel es un objeto de la clase definida en el Modelo, llamado Usuario. Los *TextBox* asociados a cada campo están enlazados a las propiedades correspondiente de dicho objeto.

Por último, se define un comando asociado al botón *Limpiar*, que permitirá reiniciar los datos.

6. Ejemplo de MVVM - Vista

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        this.DataContext = new MainWindowVM();
    }

    private void LimpiarCommand_Executed(object sender, ExecutedRoutedEventArgs e)
    {
        (this.DataContext as MainWindowVM).Limpiar();
    }
}
```


En esta diapositiva se muestra el code behind de la Vista. Por un lado, en el constructor, se crea un nuevo objeto de la clase Vista Modelo (incluida en la diapositiva siguiente) y se asocia como DataContext de la Ventana.

Por otro lado, el code behind contiene los métodos asociados a los comandos. Pero como podemos apreciar, no se implementa la lógica del comando, si no que se invoca el método correspondiente de la Vista Modelo, que implementará dicha lógica.

6. Ejemplo de MVVM – Vista Modelo

```
class MainWindowVM : INotifyPropertyChanged
{
    public Persona Usuario { get; set; }

    public MainWindowVM()
    {
        Usuario = new Persona();
    }

    public void Limpiar()
    {
        Usuario = new Persona();
    }

    public event PropertyChangedEventHandler PropertyChanged;
}
```


Por último, podemos ver el código de la Vista Modelo. Como podemos ver, la Vista Modelo incluye los datos necesarios para la Vista (en este caso, un objeto de la clase Persona) y la implementación de la lógica de presentación (el método Limpiar).

La correcta implementación del patrón MVVM en aplicaciones grandes suele conllevar la aparición de mucho código para implementar el patrón. Para intentar reducir este código y facilitar el uso de MVVM, han surgido diferentes frameworks, como los que se indican en la diapositiva:

- Prism Library
- MVVM Light Toolkit
- Caliburn.Micro
- MVVM Cross